# **Python Dictionaries**

thisdict = {  
   "brand": "Ford",  
   "model": "Mustang",  
   "year": 1964  
 }

## **Dictionary**

Dictionaries are used to store data values in key:value pairs.

A dictionary is a collection which is ordered\*, changeable and do not allow duplicates.

As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

Dictionaries are written with curly brackets, and have keys and values:

### **Example**

Create and print a dictionary:

thisdict ={  
   "brand": "Ford",  
   "model": "Mustang",  
   "year": 1964  
 }  
 print(thisdict)

## **Dictionary Items**

Dictionary items are ordered, changeable, and does not allow duplicates.

Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

### **Example**

Print the "brand" value of the dictionary:

thisdict ={  
   "brand": "Ford",  
   "model": "Mustang",  
   "year": 1964  
 }  
 print(thisdict["brand"])

## **Ordered or Unordered?**

As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change.

Unordered means that the items does not have a defined order, you cannot refer to an item by using an index.

## **Changeable**

Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.

## **Duplicates Not Allowed**

Dictionaries cannot have two items with the same key:

### **Example**

Duplicate values will overwrite existing values:

thisdict ={  
   "brand": "Ford",  
   "model": "Mustang",  
   "year": 1964,  
  "year": 2020  
 }  
 print(thisdict)

## **Dictionary Length**

To determine how many items a dictionary has, use the len() function:

### **Example**

Print the number of items in the dictionary:

print(len(thisdict))

## **Dictionary Items - Data Types**

The values in dictionary items can be of any data type:

### **Example**

String, int, boolean, and list data types:

thisdict ={  
   "brand": "Ford",  
   "electric": False,  
   "year": 1964,  
  "colors": ["red", "white", "blue"]  
 }

## **type()**

From Python's perspective, dictionaries are defined as objects with the data type 'dict':

<class 'dict'>

### **Example**

Print the data type of a dictionary:

thisdict ={  
   "brand": "Ford",  
   "model": "Mustang",  
   "year": 1964  
 }  
 print(type(thisdict))

## **Python Collections (Arrays)**

There are four collection data types in the Python programming language:

* [**List**](https://www.w3schools.com/python/python_lists.asp) is a collection which is ordered and changeable. Allows duplicate members.
* [**Tuple**](https://www.w3schools.com/python/python_tuples.asp) is a collection which is ordered and unchangeable. Allows duplicate members.
* [**Set**](https://www.w3schools.com/python/python_sets.asp) is a collection which is unordered, unchangeable\*, and unindexed. No duplicate members.
* **Dictionary** is a collection which is ordered\*\* and changeable. No duplicate members.

\*Set *items* are unchangeable, but you can remove and/or add items whenever you like.

\*\*As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

# **Python - Access Dictionary Items**

## **Accessing Items**

You can access the items of a dictionary by referring to its key name, inside square brackets:

### **Example**

Get the value of the "model" key:

thisdict ={  
   "brand": "Ford",  
   "model": "Mustang",  
   "year": 1964  
 }  
x = thisdict["model"]

There is also a method called get() that will give you the same result:

### **Example**

Get the value of the "model" key:

x = thisdict.get("model")

## **Get Keys**

The keys() method will return a list of all the keys in the dictionary.

### **Example**

Get a list of the keys:

x = thisdict.keys()

The list of the keys is a *view* of the dictionary, meaning that any changes done to the dictionary will be reflected in the keys list.

### **Example**

Add a new item to the original dictionary, and see that the keys list gets updated as well:

car = {  
"brand": "Ford",  
"model": "Mustang",  
"year": 1964  
}  
   
x = car.keys()  
  
print(x) #before the change  
  
car["color"] = "white"  
  
print(x) #after the change

## **Get Values**

The values() method will return a list of all the values in the dictionary.

### **Example**

Get a list of the values:

x = thisdict.values()

The list of the values is a *view* of the dictionary, meaning that any changes done to the dictionary will be reflected in the values list.

### **Example**

Make a change in the original dictionary, and see that the values list gets updated as well:

car = {  
"brand": "Ford",  
"model": "Mustang",  
"year": 1964  
}  
   
x = car.values()  
  
print(x) #before the change  
  
car["year"] = 2020  
  
print(x) #after the change

### **Example**

Add a new item to the original dictionary, and see that the values list gets updated as well:

car = {  
"brand": "Ford",  
"model": "Mustang",  
"year": 1964  
}  
   
x = car.values()  
  
print(x) #before the change  
  
car["color"] = "red"  
  
print(x) #after the change

## **Get Items**

The items() method will return each item in a dictionary, as tuples in a list.

### **Example**

Get a list of the key:value pairs

x = thisdict.items()

The returned list is a *view* of the items of the dictionary, meaning that any changes done to the dictionary will be reflected in the items list.

### **Example**

Make a change in the original dictionary, and see that the items list gets updated as well:

car = {  
"brand": "Ford",  
"model": "Mustang",  
"year": 1964  
}  
   
x = car.items()  
  
print(x) #before the change  
  
car["year"] = 2020  
  
print(x) #after the change

### **Example**

Add a new item to the original dictionary, and see that the items list gets updated as well:

car = {  
"brand": "Ford",  
"model": "Mustang",  
"year": 1964  
}  
   
x = car.items()  
  
print(x) #before the change  
  
car["color"] = "red"  
  
print(x) #after the change

## **Check if Key Exists**

To determine if a specified key is present in a dictionary use the in keyword:

### **Example**

Check if "model" is present in the dictionary:

thisdict ={  
   "brand": "Ford",  
   "model": "Mustang",  
   "year": 1964  
 }  
 if "model" in thisdict:  
  print("Yes, 'model' is one of the keys in the thisdict dictionary")

# **Python - Change Dictionary Items**

## **Change Values**

You can change the value of a specific item by referring to its key name:

### **Example**

Change the "year" to 2018:

thisdict ={  
   "brand": "Ford",  
   "model": "Mustang",  
   "year": 1964  
 }  
thisdict["year"] = 2018

## **Update Dictionary**

The update() method will update the dictionary with the items from the given argument.

The argument must be a dictionary, or an iterable object with key:value pairs.

### **Example**

Update the "year" of the car by using the update() method:

thisdict ={  
   "brand": "Ford",  
   "model": "Mustang",  
   "year": 1964  
 }  
thisdict.update({"year": 2020})

# **Python - Add Dictionary Items**

## **Adding Items**

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

### **Example**

thisdict ={  
   "brand": "Ford",  
   "model": "Mustang",  
   "year": 1964  
 }  
thisdict["color"] = "red"  
print(thisdict)

## **Update Dictionary**

The update() method will update the dictionary with the items from a given argument. If the item does not exist, the item will be added.

The argument must be a dictionary, or an iterable object with key:value pairs.

### **Example**

Add a color item to the dictionary by using the update() method:

thisdict ={  
   "brand": "Ford",  
   "model": "Mustang",  
   "year": 1964  
 }  
thisdict.update({"color": "red"})

# **Python - Remove Dictionary Items**

## **Removing Items**

There are several methods to remove items from a dictionary:

### **Example**

The pop() method removes the item with the specified key name:

thisdict ={  
   "brand": "Ford",  
   "model": "Mustang",  
   "year": 1964  
 }  
thisdict.pop("model")  
 print(thisdict)

### **Example**

The popitem() method removes the last inserted item (in versions before 3.7, a random item is removed instead):

thisdict ={  
   "brand": "Ford",  
   "model": "Mustang",  
   "year": 1964  
 }  
thisdict.popitem()  
 print(thisdict)

### **Example**

The del keyword removes the item with the specified key name:

thisdict ={  
   "brand": "Ford",  
   "model": "Mustang",  
   "year": 1964  
 }  
del thisdict["model"]  
print(thisdict)

### **Example**

The del keyword can also delete the dictionary completely:

thisdict ={  
   "brand": "Ford",  
   "model": "Mustang",  
   "year": 1964  
 }  
del thisdict  
print(thisdict) #this will cause an error because "thisdict" no longer exists.

### **Example**

The clear() method empties the dictionary:

thisdict ={  
   "brand": "Ford",  
   "model": "Mustang",  
   "year": 1964  
 }  
thisdict.clear()  
print(thisdict)

# **Python - Loop Dictionaries**

## **Loop Through a Dictionary**

You can loop through a dictionary by using a for loop.

When looping through a dictionary, the return value are the *keys* of the dictionary, but there are methods to return the *values* as well.

### **Example**

Print all key names in the dictionary, one by one:

for x in thisdict:  
  print(x)

### **Example**

Print all *values* in the dictionary, one by one:

for x in thisdict:  
  print(thisdict[x])

### **Example**

You can also use the values() method to return values of a dictionary:

for x in thisdict.values():  
  print(x)

### **Example**

You can use the keys() method to return the keys of a dictionary:

for x in thisdict.keys():  
  print(x)

### **Example**

Loop through both *keys* and *values*, by using the items() method:

for x, y in thisdict.items():  
  print(x, y)

# **Python - Copy Dictionaries**

## **Copy a Dictionary**

You cannot copy a dictionary simply by typing dict2 = dict1, because: dict2 will only be a *reference* to dict1, and changes made in dict1 will automatically also be made in dict2.

There are ways to make a copy, one way is to use the built-in Dictionary method copy().

### **Example**

Make a copy of a dictionary with the copy() method:

thisdict ={  
   "brand": "Ford",  
   "model": "Mustang",  
   "year": 1964  
 }  
mydict = thisdict.copy()  
 print(mydict)

Another way to make a copy is to use the built-in function dict().

### **Example**

Make a copy of a dictionary with the dict() function:

thisdict ={  
   "brand": "Ford",  
   "model": "Mustang",  
   "year": 1964  
 }  
mydict = dict(thisdict)  
 print(mydict)

# **Python - Nested Dictionaries**

## **Nested Dictionaries**

A dictionary can contain dictionaries, this is called nested dictionaries.

### **Example**

Create a dictionary that contain three dictionaries:

myfamily = {  
  "child1" : {  
    "name" : "Emil",  
 "year" : 2004  
  },  
  "child2" : {  
    "name" : "Tobias",  
    "year" : 2007  
  },  
  "child3" : {  
    "name" : "Linus",  
 "year" : 2011  
  }  
}

Or, if you want to add three dictionaries into a new dictionary:

### **Example**

Create three dictionaries, then create one dictionary that will contain the other three dictionaries:

child1 = {  
  "name" : "Emil",  
  "year" : 2004  
}  
child2 = {  
  "name" : "Tobias",  
  "year" : 2007  
}  
child3 = {  
  "name" : "Linus",  
  "year" : 2011  
}  
  
myfamily = {  
  "child1" : child1,  
  "child2" : child2,  
  "child3" : child3  
}

# **Python Dictionary Methods**

## **Dictionary Methods**

Python has a set of built-in methods that you can use on dictionaries.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [clear()](https://www.w3schools.com/python/ref_dictionary_clear.asp) | Removes all the elements from the dictionary |
| [copy()](https://www.w3schools.com/python/ref_dictionary_copy.asp) | Returns a copy of the dictionary |
| [fromkeys()](https://www.w3schools.com/python/ref_dictionary_fromkeys.asp) | Returns a dictionary with the specified keys and value |
| [get()](https://www.w3schools.com/python/ref_dictionary_get.asp) | Returns the value of the specified key |
| [items()](https://www.w3schools.com/python/ref_dictionary_items.asp) | Returns a list containing a tuple for each key value pair |
| [keys()](https://www.w3schools.com/python/ref_dictionary_keys.asp) | Returns a list containing the dictionary's keys |
| [pop()](https://www.w3schools.com/python/ref_dictionary_pop.asp) | Removes the element with the specified key |
| [popitem()](https://www.w3schools.com/python/ref_dictionary_popitem.asp) | Removes the last inserted key-value pair |
| [setdefault()](https://www.w3schools.com/python/ref_dictionary_setdefault.asp) | Returns the value of the specified key. If the key does not exist: insert the key, with the specified value |
| [update()](https://www.w3schools.com/python/ref_dictionary_update.asp) | Updates the dictionary with the specified key-value pairs |
| [values()](https://www.w3schools.com/python/ref_dictionary_values.asp) | Returns a list of all the values in the dictionary |

# **Python Dictionary Exercises**

## **Test Yourself With Exercises**

Now you have learned a lot about dictionaries, and how to use them in Python.

Are you ready for a test?

Try to insert the missing part to make the code work as expected:

## **Test Yourself With Exercises**

## **Exercise:**

Use the get method to print the value of the "model" key of the car dictionary.

car = { "brand": "Ford", "model": "Mustang", "year": 1964 } print()